
# Stroboscope: Declarative Network Monitoring on a Budget

**Olivier Tilmans,** *Université Catholique de Louvain;* **Tobias Bühler,** *ETH Zürich;*
**Ingmar Poese,** *BENOCS;* **Stefano Vissicchio,** *University College London;*
**Laurent Vanbever,** *ETH Zürich*

https://www.usenix.org/conference/nsdi18/presentation/tilmans

**This paper is included in the Proceedings of the 15th USENIX Symposium on Networked Systems Design and Implementation (NSDI '18).**

**April 9–11, 2018 • Renton, WA, USA**

ISBN 978-1-931971-43-0

# Stroboscope: Declarative Network Monitoring on a Budget

**https://stroboscope.ethz.ch**

Olivier Tilmans
*Université catholique de Louvain*

Tobias Bühler
*ETH Zürich*

Ingmar Poese
*BENOCS*

Stefano Vissicchio
*University College London*

Laurent Vanbever
*ETH Zürich*

## Abstract

For an Internet Service Provider (ISP), getting an accurate picture of how its network behaves is challenging. Indeed, given the carried traffic volume and the impossibility to control end-hosts, ISPs often have no other choice but to rely on heavily sampled traffic statistics, which provide them with coarse-grained visibility at a less than ideal time resolution (seconds or minutes).

We present Stroboscope, a system that enables fine-grained monitoring of *any* traffic flow by instructing routers to mirror millisecond-long traffic slices in a programmatic way. Stroboscope takes as input high-level monitoring queries together with a budget and automatically determines: *(i)* which flows to mirror; *(ii)* where to place mirroring rules, using fast and provably correct algorithms; and *(iii)* when to schedule these rules to maximize coverage while meeting the input budget.

We implemented Stroboscope, and show that it scales well: it computes schedules for large networks and query sizes in few seconds, and produces a number of mirroring rules well within the limits of current routers. We also show that Stroboscope works on existing routers and is therefore immediately deployable.

## 1 Introduction

Not all networks are created equally when it comes to monitoring. ISP networks, in particular, suffer from extremely poor visibility. As they do not control end-hosts and carry huge amounts of traffic, ISP operators often have no choice but to rely on pure in-network solutions based on random packet sampling (i.e., NetFlow [1] or sFlow [2]). By design, random sampling provides no guarantee on which traffic flows will be sampled, by which router and at what time. Except for few heavy-hitters [3], even minutes-long collections of random samples typically provide coarse-grained and inaccurate bandwidth estimations for the large majority of the prefixes. Moreover, the likelihood of randomly sampling the same flow across the network is extremely low; hence, it is basically impossible to use random samples for reasoning on the network-wide forwarding behavior, and monitoring anything else than bandwidth.

We confirmed these limitations in an actual Tier-1 ISP by analyzing the Netflow data collected by hundreds of routers over 10 minutes. We observed that most BGP prefixes (65%) are not observed at all, 15% of them are observed only twice, and just 10% of all prefixes are observed more than 30 times. Even worse, 75% of these observed flows were only seen on a single router, making it *impossible to track flows network-wide*, even for the largest heavy hitters.

As a result, ISP operators are currently incapable of answering practical questions like: *What is the ingress router for a given packet seen at a specific node? Which paths does the traffic follow? Is the network-wide latency acceptable? Is traffic load-balanced as expected?*

**Stroboscope** This paper presents Stroboscope, a scalable monitoring system that complements existing tools like NetFlow, by enabling fine-grained monitoring of *any* traffic flow. Stroboscope exploits the possibility to extract small traffic samples (i.e., slices) in a programmatic way, by activating and deactivating traffic mirroring for any destination prefix, up to a single IP address, network-wide, and within milliseconds. Our tests confirm that this possibility is available today, on currently deployed routers, making Stroboscope immediately deployable.

By coordinating packet mirroring across routers, Stroboscope implements *deterministic packet sampling*: it collects copies of the same packets from multiple locations, following such packets as they cross the network. This enables Stroboscope to precisely measure the network forwarding behavior including traffic paths, one-way delays and load-balancing ratios. Traffic slices with no packets are also informative: Stroboscope uses them to determine additional forwarding properties, like packet loss and devices not receiving specific flows.

**Challenges** Given a high-level query, determining which flows to mirror, where and when is both hard and potentially dangerous—especially when considering arbitrary network dynamics (e.g., unexpected traffic shifts). Aggressive mirroring strategies can lead to significant congestion (e.g., if many routers mirror traffic for popular destinations) and inaccurate results (e.g., if congestion affects the mirrored traffic). Conversely, conservative strategies can lead to poor coverage and slow answers.

**Compilation** Stroboscope tackles those challenges on behalf of operators. From high-level queries, it automatically derives how to mirror traffic so as to maximize monitoring accuracy without exceeding a budget, while also adapting to network dynamics in near real time.

Stroboscope's compilation process follows three steps. First, Stroboscope decides *what* (which prefixes) to mirror for every query, dynamically adapting this decision according to the amount of mirrored packets. Second, Stroboscope computes *where* to activate mirroring rules, in order to maximize coverage while minimizing the impact on the budget. Third, Stroboscope calculates *when* to mirror and for how long, producing a budget-compliant schedule, optimized across all input queries.

**Guarantees** Stroboscope provides strong guarantees in terms of budget compliance, even in the presence of unpredictable network dynamics. In fact, traffic mirrored by Stroboscope can only exceed the budget for at most the few milliseconds needed to collect a single traffic slice.

**Implementation** We implemented Stroboscope, and show that it scales well: it computes schedules for large networks and query sizes in few seconds, and produces a number of mirroring rules well within the limits of current routers. We also demonstrate how to build practical monitoring applications on top of Stroboscope, such as estimating one-way delays, loss rates, or load-balancing ratios for any destination prefix.

**Contributions** Our earlier work [4] showed the benefits of mirroring thin traffic slices to monitor networks. This paper goes further by describing the complete design, implementation and evaluation of the corresponding system. We make the following contributions:

- A novel fine-grained and scalable monitoring approach based on deterministic traffic sampling (§2);
- Practical algorithms to: *(i)* estimate unknown traffic demands in real time (§3); and *(ii)* compute optimally placed mirroring rules (§4), as well as schedule them while adhering to a given budget (§5);
- A full implementation of Stroboscope (§6) along with a thorough evaluation using benchmarks, simulations and tests on Cisco routers (§7);
- A case study demonstrating how to use Stroboscope measurements to estimate one-way delays, loss rates, and load-balancing ratios (§8).

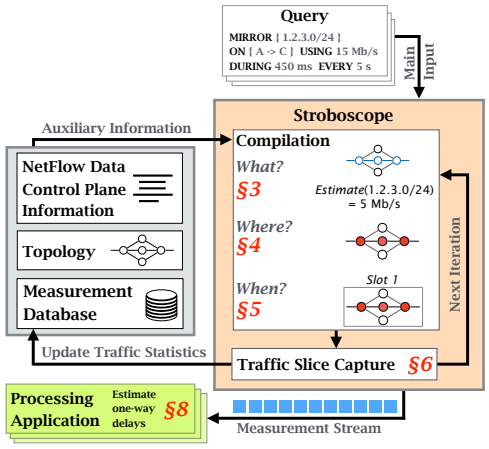



**Figure 1: Stroboscope translates high-level queries to measurement streams by capturing packet slices.**

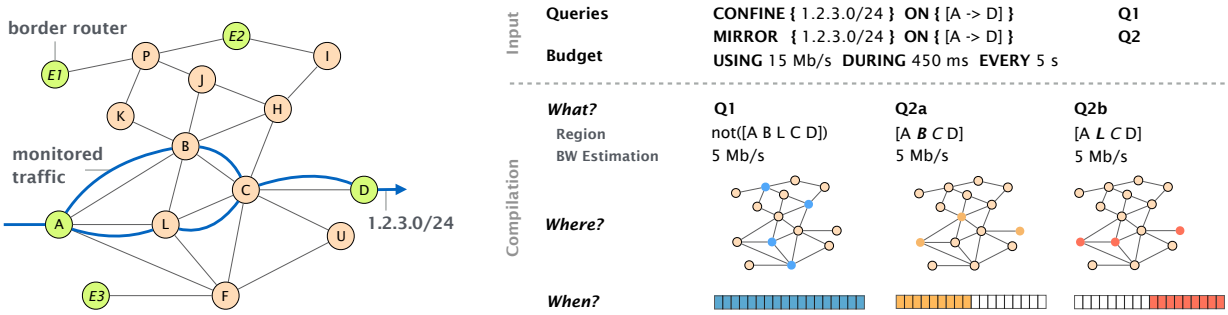
## 2 Overview

In this section, we provide an intuitive description of Stroboscope (see Fig. 1) using a running example. Specifically, we consider a network operator who receives complaints from customers trying to reach one prefix (1.2.3.0/24) through its infrastructure (see Fig. 2a). Following up on the complaints, the operator wants to: *(i)* check that the corresponding traffic follows the expected paths; and *(ii)* measure key performance indicators, such as packet loss rate and path latencies.

**Specifying queries** Stroboscope allows operators to define their monitoring goals using an SQL-like language:

```
((MIRROR | CONFINE) <prefixes>
    ON <paths> )+
 USING <Gbps> DURING <sec> EVERY <sec>
```

These monitoring queries specify for which IP prefixes, up to a single IP address, traffic should be mirrored (`MIRROR`) or confined (`CONFINE`), and where (`ON`), e.g., on a specific node, along a specified path or following the ones computed by the routing protocols (indicated with the `->` operator, e.g. *A* `->` *D*). `MIRROR` and `CONFINE` queries differ in when they mirror traffic: the former continuously mirrors traffic while the latter only mirrors traffic that *leaves* a specified region. In addition, operators can specify constraints on: *(i)* the maximum rate of mirrored traffic (`USING`) allowed; *(ii)* the duration of any measurement campaign (`DURING`); and *(iii)* the frequency at which to run measurements (`EVERY`).
Coming back to the example above, the operator can instruct Stroboscope to mirror traffic along all IGP paths between *A* and *D* using a `MIRROR` and a `->` construct (see Fig. 2b). Additionally, she can use a `CONFINE` construct to verify that these paths are the only ones carrying traffic towards 1.2.3.0/24.




(a) Sample network.

(b) Compilation process translating high-level queries in a measurement campaign.

**Figure 2: From high-level monitoring queries and the current network state, Stroboscope computes a measurement campaign schedule meeting the monitoring budget.**

While simple, our language supports several practical use cases. Among others, `MIRROR` queries enable network-wide path tracing, i.e., following a given packet as it traverses a sequence of nodes. Packet copies can then be analyzed by monitoring applications to estimate data-plane performance, like packet loss or path latency, or to inspect packet payloads. `CONFINE` queries are especially useful to detect unwanted forwarding behavior (e.g., traffic shifts, security policies) at runtime, and to complement information from `MIRROR` queries (e.g., on paths not taken by given traffic flows).

**A three-staged compilation process** From high-level queries, Stroboscope derives *measurement campaigns*, i.e., schedules of mirroring rule (de-)activations that: *(i)* provide strong guarantees on budget compliance; *(ii)* maximize accuracy by activating mirroring rules as often as possible; *(iii)* minimize the number of mirroring locations to both lower the mirrored traffic volume and decrease the control-plane overhead. Stroboscope derives these measurement campaigns in *three stages*.

**Stage 1: Resolving high-level queries (§3)** First, Stroboscope translates any input query into a concrete one defined on actual paths and flows. To this end, it collects routing (e.g., IGP and BGP) feeds and NetFlow records whenever available. It also maintains a *measurement database*, storing results from past monitoring campaigns. Based on this information, Stroboscope estimates per-prefix traffic volumes and computes their forwarding paths. In our example, Stroboscope estimates the traffic demand for 1.2.3.0/24 to be 5 Mbps and resolves *[A -> D]* (Q2) in two sub-queries {*[A B C D]*,*[A L C D]*} (Q2a, Q2b), one for each path.

**Stage 2: Optimizing mirroring locations (§4)** Second, Stroboscope minimizes the number of mirroring rules by optimizing their locations using two provably correct algorithms. Doing so, it reduces the mirrored traffic and the control-plane overhead to activate them.

The first algorithm (§4.1) optimizes the placement of `MIRROR` queries, like Q2a and Q2b in Fig. 2b. The key insight is to leverage properties of the complete network topology to prune mirroring rules. For instance, for Q2a, no mirroring rule is required on router $C$, as $C$ is the only 1-hop path between $B$ and $D$. By observing the TTL of mirrored packets at $B$ and $D$, we can therefore be sure that traffic traversed $C$, without actually mirroring there.

The second algorithm (§4.2) deals with `CONFINE` queries, like Q1. The key insight is to place heavily rate-limited mirroring rules, all around the region specified in the query. This way, no packets are mirrored for correct `CONFINE` queries, and few packets per location are mirrored for incorrect queries. Our algorithm optimizes the position of surrounding rules, as exemplified in Fig. 2b. For example, the algorithm places only one mirroring rule on $P$ to detect possible packets crossing *[A B]* and leaving the network at $E1$ or $E2$.

**Stage 3: Computing measurement campaigns (§5)** Third, Stroboscope schedules mirroring rules over time. These schedules use the estimated traffic volumes to meet the budget, while packing as many measurements as possible to increase monitoring accuracy. Computing such schedules is a variant of the bin-packing problem, which is $NP$-hard. To scale, Stroboscope encompasses fast approximation heuristics ($O(n \log n)$ where $n$ is the number of queries) whose results are close to optimal. Our scheduling approach enforces deterministic sampling: packets for one specific query are mirrored from well-defined locations for a given amount of time.

In our example, Q2a and Q2b in Fig. 2 cannot be scheduled at the same time given the specified budget of 15 Mbps. Indeed, with 4 different mirroring rules, they would require a total of 20 Mbps. Stroboscope therefore schedules Q2a and Q2b each for half of the timeslots. In addition, as Q1 does not mirror any traffic unless a violation is detected, Stroboscope schedules Q1 for all the timeslots, so that any violation to Q1 can be detected.

**Budget Guarantees** The ability of a Stroboscope schedule to meet the budget requirements inherently depends on two assumptions, both checked at runtime. First, Stroboscope checks its demand estimations by monitoring the total traffic being mirrored and stops the measurement campaign when detecting a budget violation. Such a premature termination is enforced within one mirroring timeslot (a few milliseconds). Second, as **CONFINE** queries are not expected to mirror traffic, and only require one packet when violated, they are rate-limited.

**Outputs** Stroboscope's runtime (§6) carries out measurement campaigns instructing routers to mirror query-defined traffic flows for a specific amount of time. Stroboscope outputs a stream of collected packets with their meta-data (e.g., timestamp, corresponding query), meant to be processed by the operators or external applications.

## 3 From Abstract to Concrete Queries

Given an input query, the first operation performed by Stroboscope is to concretely define the prefix and the region to monitor. We now detail how this happens.

**Resolving loosely defined regions** In Stroboscope's input queries (like Q2 in Fig. 2b), regions to monitor can be specified using the `->` operator. Stroboscope replaces any expression *s*`->`*t* with the forwarding paths from router *s* to router *t* as provided by the routing protocols (e.g., the IGP) running in the network. If no IGP path can be found, Stroboscope returns a compilation error. For example, in Fig. 2, *[A* `->` *D]* will be translated into *[A B C D]* and *[A L C D]* if those are all the IGP forwarding paths from *A* to *D* for 1.2.3.0/24.

Whenever the `->` operator is present at the start (resp. end) of a query, the Stroboscope replaces it with the set of all ingress (resp. egress) routers that receive traffic for the prefix in the query—e.g., leveraging BGP information if present, or static knowledge of all network border routers. Using this feature, the queries from Fig. 2b can be generalized to all paths terminating in *D*: it would be sufficient to replace *[A* `->` *D]* with *[* `->` *D]* in the queries, *discovering* on the fly which ingresses are active. Those translations are updated at the start of each measurement campaign, so that Stroboscope performs the following measurements consistently with the latest available routing information, and flags the previous measurements if collected during routing changes.

**Estimating traffic volumes** In order to match the budget, Stroboscope needs information about traffic volumes for every prefix specified in the input queries. It is *fundamentally impossible* to exactly know how much traffic will be destined to any prefix ahead of measurements: in theory, any flow can unpredictably vary over time. Stroboscope does not require traffic estimation to be 100% accurate, as it includes runtime mechanisms to bound the amount of excessive traffic (see §5.2). Yet, for Stroboscope to avoid computing infeasible schedules, we would like traffic estimation to be as close to the real demands as possible. To this end, Stroboscope implements a dynamic traffic estimation technique, based on data collected during past measurement campaigns. For each prefix involved in any input query, the measurement database stores the maximum demand measured by Stroboscope over a customizable number of minutes (5, by default). Stroboscope then uses such value as a conservative estimate of the traffic that will be received for that prefix during the next iteration.

The above procedure is applicable if Stroboscope has historical data for all the queried prefixes. This condition might not hold in several cases, e.g., for prefixes not recently mirrored and those in **CONFINE** queries (for which no or few packets are mirrored, as discussed in §2). Stroboscope solves the absence of historical data in two ways. First, it can infer estimates from sampled traffic (e.g., as collected by NetFlow). In this case, Stroboscope sets the peak value recorded by random sampling as initial traffic estimation for the prefixes tracked in a significant number of samples (e.g., more than 30 in 5 minutes). This way, it exploits random sampling for what it is good at: bandwidth estimation for heavy hitters [3]. Second, for all the prefixes not covered by enough random sampling data, Stroboscope runs a specific, bootstrapping measurement campaign to estimate their traffic volume. In particular, Stroboscope reserves one minimal timeslot per prefix, and activates mirroring on all the routers in the region specified by the query (e.g., on all routers in *[A, B, C, D]* for Q2a in Fig. 2). If no traffic is captured, more timeslots are reserved to the same prefix.

We stress that the risk of significantly exceeding the budget by running bootstrapping campaigns is limited. First, those measurements are targeted to prefixes that are likely to carry a limited amount of traffic since they generated few or no observations over minutes of random sampling. In addition, traffic for each prefix is mirrored for a minimal timeslot, which would last about 25 ms in our current implementation (see §7).

Stroboscope also outputs packets collected during bootstrapping campaigns with convenient meta-data. This enables operators and special-purpose applications to select sub-prefixes of the queried destinations that best match the query purpose.

## 4 Optimizing Mirroring Locations

Stroboscope runs distinct algorithms to select mirroring locations for **MIRROR** (§4.1) and **CONFINE** (§4.2) queries. These algorithms minimize the number of mirroring locations while also providing *high accuracy guarantees*

of the produced measurements (e.g., packets violating **CONFINE** queries are never missed). Reducing the mirroring locations let Stroboscope: *(i)* answer more queries at the same time within the input budget; and *(ii)* decrease the control-plane overhead by changing less mirroring rules during measurement campaigns.

Stroboscope's algorithms take as input the operator-specified queries and the complete network topology, including all currently down links and nodes. Considering all possible links and nodes ensures that the algorithms always guard against all possible network paths, and never select mirroring locations breaking the accuracy guarantees due to transient topology changes.

## 4.1 Key-points Sampling algorithm

We developed the key-points sampling (KPS) algorithm. Stroboscope uses the KPS algorithm to select mirroring locations for **MIRROR** queries.

**Goal** Given a **MIRROR** query on a path $P$, KPS selects a set of routers which will capture traffic crossing $P$, while also enabling to distinguish packets forwarded outside $P$, which would violate the query.

Note that this goal cannot be achieved through the naive solution of mirroring only at the extremes of the path. For example, if a **MIRROR** query is defined on path *[A B C D]* in Fig. 2a would only place mirroring rules on $A$ and $D$, packets forwarded over *[A B C D]* would be indistinguishable from those flowing over *[A L C D]*.

**General solution** By default, KPS returns all the routers in the path. This guarantees that the resulting measurement campaigns track all packets crossing any subset of routers in the path. For each mirrored packet, Stroboscope checks if there exists a sequence of routers such that the Time-To-Live (TTL) of the packet is decreased exactly by 1 at each hop in the sequence. Assuming that every router decreases packets' TTL by 1[1], a mirrored packet must have followed the path in the query if and only if Stroboscope finds such a sequence for that packet.

**Optimizations** KPS goes beyond this general solution whenever mirroring locations can be reduced according to the complete network topology. To this end, it exploits the following theorem, proven in Appendix A.1.

**Theorem 1.** *Let a forwarding path $P$ be the concatenation of sub-paths $Q_1, \ldots, Q_n$.* ***MIRROR*** *queries on $P$ can be correctly answered by mirroring only on the endpoints $s_i$ and $t_i$ of all $Q_i$ such that no other forwarding path from $s_i$ to $t_i$ has the same length as $Q_i$.*

As an illustration, consider Fig. 2a. The path *[A B C D]* can be seen as the concatenation of *[A B]* and *[B C D]*. Also, *[B C D]* is the only path in the topology of length 3 from $B$ to $D$. Theorem 1 states that we can skip C as mirroring location. This is intuitively true because we can distinguish packets traversing *[B C D]* as the only ones whose TTL in $D$ is equal to the TTL in $B$ minus 2.

**Algorithm** Given a path $P$, KPS checks all the concatenations of sub-paths that result in $P$. For each concatenation, KPS checks if Theorem 1 holds on each sub-path, performing a depth-first search on the network graph truncated at a depth equal to the sub-path length[2]. KPS then stores the first and last router in the sub-paths compliant with Theorem 1 plus all the routers in the other sub-paths as the set of mirroring points for that concatenation. Finally, it returns a set with minimal cardinality.

For example, for the path *[A B C D]*, KPS sequentially considers the concatenations *[A B C D]*, *[A B][B C D]*, *[A B C][C D]*, and *[A B][B C][C D]*. By following this order, the first concatenation is the minimal one, as a concatenation with $n$ elements requires at least $n+1$ mirroring locations, corresponding to the first and last routers in every sub-path. For instance, if Theorem 1 was applying to *[A B C D]*, KPS would immediately return $A$ and $D$ as mirroring locations. Instead, as *[A L C D]* has the same length than *[A B C D]*, Theorem 1 does not hold. This implies that at least 3 mirroring locations will be needed (e.g., *{A,B,D}* is the minimum set of locations for *[A B][B C D]*).

KPS is theoretically inefficient, since any of the depth-first search it runs can potentially explore an exponential number of paths. However, our evaluation (§7.1) shows that KPS takes milliseconds to process paths in real networks, due to their sparsity and the limited path lengths.

Stroboscope also supports **MIRROR** queries defined on *regions*, i.e., connected components of the network graph. Such queries are answered by creating sub-queries for all the paths in the region and applying the above procedure to each sub-query.

## 4.2 Surrounding algorithm

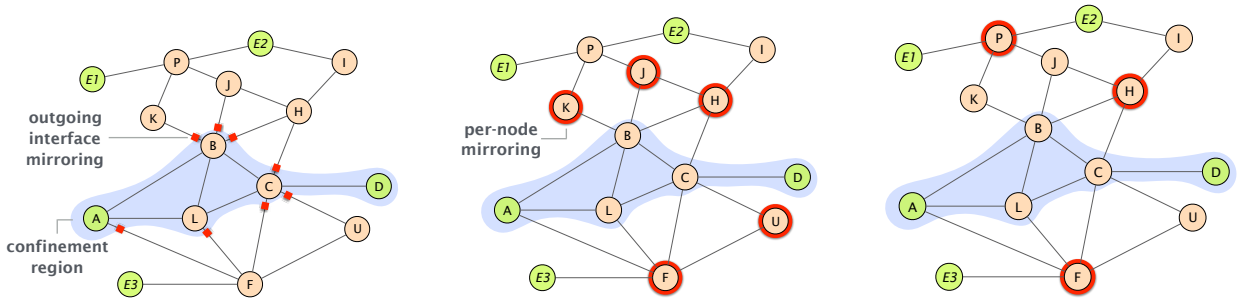
To find mirroring locations to answer **CONFINE** queries, Stroboscope runs the surrounding algorithm.

**Goal** Given a **CONFINE** query on a region $R$ (which we call *confinement region*), the surrounding algorithm selects mirroring locations (routers or network interfaces) which will mirror any packet exiting the region.

Computing these locations while complying with the above goal is trickier than what it may look like. One challenge is to avoid capturing interfering traffic, that is, packets for the prefix in the query not traversing the confinement region. In Fig. 2, for example, $P$ could not be a

[1] This assumption is consistent with the default behavior of commercial routers for both IP and MPLS packets [5, 6].

[2] The result of this check is cached, to possibly skip the depth-first search while re-evaluating the same sub-path in other concatenations.




(a) Default mirroring locations computed by the surrounding algorithm on **CONFINE** queries defined on the region *[A B L C D]*.

(b) Optimized placement if no flow for the queried destination crosses neighbors of the region *[A B L C D]* (no interfering traffic).

(c) Optimized placement in the absence of both interfering traffic and network anomalies (blackholes, forwarding loops).

**Figure 3: Depending on properties of the network graph and knowledge about the correctness of the network, we can reduce the number of mirroring locations and keep the same guarantees.**

mirroring location for Q1 if additional traffic (not shown in the figure) for 1.2.3.0/24 enters in *E1* and is forwarded on the path *[E1 P E2]*—remember that the region specified in the query may not include all and only the actual forwarding paths for a prefix.

**General solution** Given a confinement region $R$, we define the *edge surrounding* of $R$ as the set of directed edges $(r,n)$ such that $r \in R$ and $n \notin R$. By default, the surrounding algorithm returns as mirroring locations the set of outgoing interfaces of routers in $R$ that correspond to any element of the edge surrounding. Fig. 3a visualizes the output of this algorithm for Q1 in Fig. 2. The following theorem (proved in Appendix A.2) states that the default output of the surrounding algorithm is correct.

**Theorem 2.** ***CONFINE*** *queries on a region R can be correctly answered if and only if the set of mirroring locations is the edge surrounding of R.*

Intuitively, the theorem holds because: *(i)* to exit $R$, any packet must be forwarded from a router in $R$ to another outside $R$, hence over a link in the edge surrounding; and *(ii)* all the captured packets are mirrored when exiting $R$.

**First optimizations** Mirroring on links in the edge surrounding of the confinement region copies no packet if traffic is indeed confined to that specified region. Nevertheless, a minimal number of locations would reduce the control-plane overhead, as Stroboscope periodically reinstalls mirroring rules—keeping them alive while guaranteeing their autonomous deactivation (§6).

The surrounding algorithm uses routing information (when available) to reduce the number of mirroring locations. Knowing all the possible forwarding paths for the queried prefixes can indeed enable to safely push mirroring locations one hop away, from outgoing interfaces of routers in $R$ to neighboring routers. In Fig. 3a for instance, if no forwarding path for 1.2.3.0/24 (from Fig. 2a) crosses $F$, $F$ itself can be added to the set of mirroring locations and we can remove all the outgoing interfaces facing $F$—saving 2 mirroring rules.

We define the *node surrounding* of a region $R$ as the set of routers that are directly connected to at least one router in $R$. Starting from the edge surrounding of $R$, the surrounding algorithm systematically replaces links ending on any router $x$ in the node surrounding of $R$ every time $x$ is part of no forwarding path for the prefix in the query. Possibly, the entire edge surrounding is replaced by the node surrounding, as shown in Fig. 3b assuming that the region defined by $A$, $B$, $L$, $C$ and $D$ contains all forwarding paths for 1.2.3.0/24. A simple extension of Theorem 2 proves the correctness of this selection.

**Optimal solution** The surrounding algorithm further reduces the number of mirroring rules in the guaranteed absence of forwarding anomalies[3], that is, no blackholes and no forwarding loops within the monitored network.

We define a *mixed-egress path* for a region $R$ as a simple path starting from a router in $R$, traversing at least one router outside $R$ and ending in any egress point. The following theorem holds, as proved in Appendix A.3.

**Theorem 3.** *In the absence of forwarding anomalies, a* ***CONFINE*** *query on a region R can be correctly answered if and only if every mixed-egress path for R contains at least one mirroring location.* [4]

The proof of Theorem 3 is based on the fact that in general, any packet exiting a region $R$ either reaches an egress point (including those in $R$), or is dropped before. In the absence of forwarding anomalies, only the former case can happen, hence it is sufficient and necessary for mirroring locations to cover all the paths ending in an

[3] This property can for example be checked by leveraging the results of other **MIRROR** queries given as input to our system.

[4] This statement does not conflict with Theorem 2, since edge and node surroundings guarantee that the condition of Theorem 3 holds.

egress point and not entirely in $R$. Consider, for example, Fig. 3c. If nodes $P$, $H$, and $F$ mirror traffic, then no packet can exit the region *[A B L C D]* without traversing some mirroring location, or incurring a forwarding-anomaly—e.g., looping on some routers to re-enter the region, or be incorrectly discarded by an internal router.

**Algorithm** Determining the default set of mirroring locations in the presence and absence of interfering traffic mainly requires to compute edge and node surroundings, respectively: both sets can be calculated by simply iterating over all the links of the input network.

In the absence of forwarding anomaly, the surrounding algorithm returns a minimal set of locations compliant with Theorem 3. To this end, it computes a set of nodes disconnecting the input region from every egress. This is a variant of the *minimal multi-terminals cut* problem. Stroboscope solves this variant in polynomial time running the algorithm described in [7].

The algorithm in [7] requires an upper bound of the size of the cut to be computed. In our case, the cardinality of the node surrounding would provide such a bound. To further improve its efficiency, Stroboscope however computes a tighter bound by heuristically removing redundant elements from the node surrounding. It initializes the cut to the node surrounding. For every node $n$ in the current cut, Stroboscope computes a simplified graph that does not include any node in the current cut except $n$, nor any link in the confinement region. For example, when considering router $U$ in Fig. 3b, the algorithm removes $F$, $K$, $J$, $H$ (as they are in the node surrounding) and all the links in the region $\{A, B, L, C, D\}$. On this simplified graph, the algorithm computes the connected component including $n$—which is, $U, C$ in our example. If there is no path in this connected component between any node in the component and an egress point (as it is for $U$ in our example), all mixed-egress paths must include at least another router in the current cut; hence, being redundant $n$ is removed from the current cut.

# 5 Computing Measurement Campaigns

Combining the information on the prefix to monitor (§3) and the result of the location algorithms (§4), we end up with a group of mirroring rules for every query. The next step performed by Stroboscope is to schedule those groups of rules, producing measurement campaigns.

Answering a query requires to simultaneously activate all its mirroring rules during a given amount of time. Also, to maximize the accuracy of measurements across queries, different groups of rules should be packed together as much as possible, but respecting the traffic and time budget. We detail how Stroboscope computes a mirroring schedule in §5.1, and adapts it at runtime in §5.2.

## 5.1 Building a Measurement Schedule

Any schedule computed by Stroboscope is made of a finite number of timeslots, and assigns every group of mirroring rules to one or more timeslots. A *timeslot* represents an interval of time, not overlapping with any other timeslot; all the mirroring rules assigned to a timeslot $s$ must be active during the time corresponding to $s$.

To meet the traffic budget, Stroboscope assigns a *cost* to every rule, reflecting the expected rate (e.g., 5 Mb/s) of traffic mirrored when the rule is active. For every **`MIRROR`** query on a prefix $p$, the corresponding rules are expected to mirror traffic for $p$; hence, the cost assigned to such rules is equal to the traffic rate for $p$, as estimated in the query pre-processing (see §3), multiplied by the number of mirroring locations (see §4.1). The cost of any **`CONFINE`** query is set to zero. In fact, mirroring rules for a **`CONFINE`** query are heavily rate-limited, hence at most a few packets per mirroring location are mirrored in the worst case—and zero if the query is correct. Note that setting the cost of **`CONFINE`** queries to zero implies that Stroboscope always schedules these queries in all timeslots. Stroboscope's scheduling problem then consists in assigning the groups of rules corresponding to **`MIRROR`** queries to every timeslot, so that the sum of the costs of all rules scheduled at every timeslot does not exceed the traffic budget defined in the queries.

Stroboscope first derives the number of timeslots, duration and spacing from the router-to-collector latencies and the monitoring time defined in the query through the **`DURING`** keyword. Then, to scale to a large number of queries and schedule sizes, Stroboscope splits the scheduling problem in two phases (see Fig. 4): a first phase where rules are scheduled as tight as possible, in a schedule of minimal duration; and a second phase, where the minimal schedule is replicated as much as possible, to maximize the usage of the budget, hence increasing monitoring accuracy. In both phases, Stroboscope can restrict to an approximate solution (e.g., for fast inclusion of new queries), as shown in the bottom part of Fig. 4.

**Timeslot duration and spacing** Timeslot durations must be long enough to ensure that packets copied at the ingress routers of any **`MIRROR`** query can also be mirrored at the corresponding egress routers. Stroboscope derives the duration of timeslots in a schedule from: *(i)* the minimal traffic slice duration, according to the used mirroring technology (see §6 and §7.3); and *(ii)* the observed maximal latency in the network, either defined statically or estimated as shown in §8. Also, to let in-flights packets arrive at the collector at the end of a timeslot, schedules generated by Stroboscope must include spacing between consecutive timeslots. We conservatively set this spacing to the maximum router-to-collector latency.

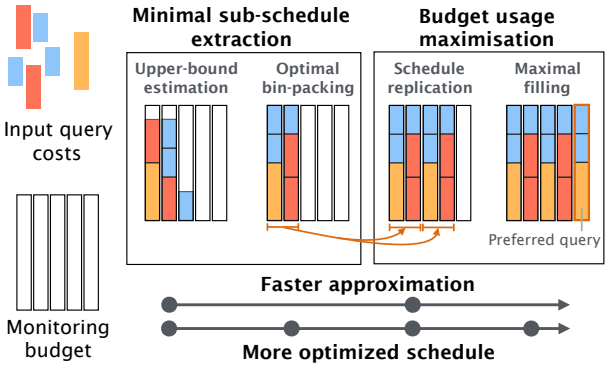



**Figure 4: Stroboscope scheduling algorithm.**

**Minimal schedule extraction** In the first phase, the scheduling algorithm assigns each group of rules to exactly one timeslot, with the goal of minimizing the number of timeslots. This is a bin packing problem, and it is therefore NP-hard. To improve time efficiency, Stroboscope first computes an upper bound on the size of the minimal schedule, using the well studied First-Fit-Decreasing heuristic—which has been proved to approximate optimum solutions with a tight bound of $\sim$ 1.22OPT [8]. The computed upper bound is then exploited to compute a minimal schedule, using a standard Integer Linear Program (ILP) formulation for bin packing problems (as detailed in Appendix B.1).

**Budget usage maximization** In the second phase, the scheduling algorithm replicates the minimal schedule as much as possible, increasing the number of timeslots allocated to all queries in an uniform way. The duration of the schedule might not be fully consumed by such replication—for example, the schedule in Fig.4 encompasses 5 timeslots, which allows to replicate its minimal schedule at most twice while wasting 1 timeslot. To fill the remaining timeslot(s), Stroboscope solves another ILP, whose objective function is to fit the maximum number of groups of rules into the input timeslots. Appendix B.2 contains a detailed description of this ILP.

## 5.2 Adapting the Schedule at Runtime

There are two possible outcomes for the scheduling just described. If Stroboscope cannot compute a schedule, it returns an error to the operator specifying the reason why the schedule could not be computed (e.g., because the time or bandwidth budget are too low). Otherwise, it starts mirroring packets according to the computed schedule. While collecting packets, it further adapts the schedule in specific cases, that we now detail.

**Guarantees on limited budget overflow** Stroboscope schedules rule activations to match the budget on the basis of traffic estimations which can be wrong (e.g., unpredictable traffic variations). While estimation errors can balance across different prefixes, using a static schedule comes with the risk of mirroring much more traffic than the budget if the predictions are greatly underestimating the actual traffic volume for some prefix.

To minimize budget overflow, Stroboscope tracks the total amount of traffic mirrored after every timeslot. Then, it compares such a total with the budget for 1 second (e.g., 1 Gb if the budget is 1 Gbps). Whenever the total mirrored traffic exceeds the 1-second budget, Stroboscope stops the ongoing measurement campaign, waits for the remaining time in the 1-second interval while computing a new schedule, and finally runs a new campaign. For example, if it detects that 1.1 Gb of traffic are mirrored in 0.7 seconds, for queries with a budget of 1 Gb/s, Stroboscope stops the measurement campaign, waits for 0.3 seconds, and then starts a new campaign.

Since the inter-timeslot spacing ensures that the collector receives all the mirrored packets before starting the next measurements (see §5.1), the runtime behavior just described yields the following property.

**Property 1.** *Stroboscope exceeds the budget in any query for at most 1 timeslot per measurement campaign.*

Note that traffic estimates are updated after the stopped campaign, so the successive campaign is much more likely not to exceed the budget again.

# 6 Implementation

We built a complete prototype of Stroboscope in $\sim 5,000$ lines of Python code, and 650 of C code[5]. Our implementation covers the entire compilation pipeline along with the logic to trigger mirroring rules on routers (Cisco or Linux-based), as well as benchmarks.

**Mirroring packets** Packet mirroring is supported by most commercial routers [9, 10]. It enables routers to duplicate packets matching given criteria (expressed using route-maps) and to send such copies to another device (e.g., over a GRE tunnel) directly in the data plane. Since packet mirroring is typically implemented in hardware, it has been experimentally shown to work at scale, with negligible CPU load and without degrading forwarding performance of mirroring routers [11].

Unfortunately, most routers only support 2 criteria to be used on all mirroring rules at the same time [9], which would prevent Stroboscope from capturing more than 2 flows per router, hence answer many real queries.

Stroboscope overcomes this limitation by indirectly triggering the mirroring of a flow, complementing mirror matching criteria with dynamic ACLs. More specifically, packet duplication primitives are pre-configured

[5] available at `https://github.com/net-stroboscope`

to match a single specific tag (e.g., a VLAN tag or a DSCP value), which we call *mirroring tag*. Stroboscope then dynamically updates ACLs to add that tag to all and only the packets to be mirrored. We use two different tag values: one for **`MIRROR`** queries, and another one for **`CONFINE`** queries which is heavily rate-limited. As **`CONFINE`** queries only need a single packet from a flow to report a violation, this mitigates the increase of mirrored traffic without losing information.

Our implementation activates mirroring rules by executing a pre-loaded script on each router, as readily possible in commercial routers (see, e.g., [12, 13]). The script takes two arguments: *(i)* a list of flows; and *(ii)* a mirroring duration. When invoked, it dynamically configures the ACL to tag all the packets in the input flow list. It then sets a timer based on the provided mirroring duration. On its expiration, it removes the configured ACL, deactivating the mirroring process. This technique requires only a *single interaction* between Stroboscope and the router, and *no separate deactivation message*. The deactivation after the predefined time interval is guaranteed.

Mechanisms like configuring ACLs through BGP Flowspec [14] or Netconf [15], or directly programming the IGP [16] to switch between VLANs [4], can all be used in Stroboscope, instead of our current in-router scripting approach. However, such alternatives impose a bigger overhead and cannot guarantee a slice duration (as the mirroring process has to be stopped remotely). We experimentally confirmed that our implementation can activate a large number of mirroring rules in a short amount of time (e.g., consistently with [17]), and evaluated the control over the slice duration in §7.3.

**Processing mirrored packets** For each mirrored packet, Stroboscope's implementation extracts: *(i)* the router ID originating it; *(ii)* its original destination IP; and *(iii)* the NIC timestamp at which it was received. At the end of each timeslot, Stroboscope outputs the collected traffic slices (possibly empty), grouped by queries, with all meta-data associated to the mirrored packets. Furthermore, it includes if packets were following the expected paths, and which packets match others.

# 7 Evaluation

We now evaluate our implementation of Stroboscope. First, we start by evaluating the algorithmic pipeline using synthetic benchmarks on realistic ISP topologies, to confirm that: *(i)* it can compute measurement campaigns in a timeframe suitable for online use; and *(ii)* it is able to maximize the accuracy of each query. We observe that the placement algorithms (§7.1) optimize mirroring locations in milliseconds, and reduces the number of mirroring rules by up to 50%. While the scheduling algorithm (§7.2) approximates schedules in milliseconds, optimized schedules increase accuracy by 15% for half the experiments. Second, we present measurements on real routers (§7.3) which confirm their ability to capture traffic slices as small as 23 ms. Finally, we validate the ability of Stroboscope to react to unexpected traffic changes within one timeslot using Mininet [18] (§7.4).

## 7.1 Placement algorithms performance

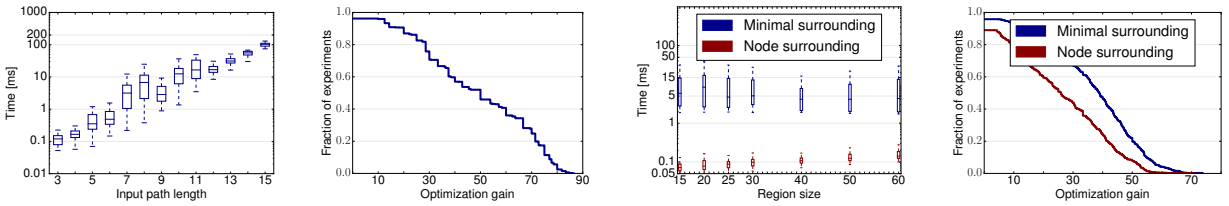
Fig. 5 shows execution time and mirroring location reduction of the placement algorithms (see §4) when run on all Rocketfuel topologies [19] and on the largest topologies from the Internet Topology Zoo [20]. We performed more than $4,000$ experiments for each algorithm. We remind that the speed of the algorithms affects Stroboscope's ability to recompute a new schedule online (i.e., to react to routing changes). However, reducing mirroring locations enables to increase the number of timeslots per schedule, hence the accuracy of query answers.

**Key-points sampling (§4.1)** We evaluate the algorithm by defining monitoring paths as random shortest-paths (according to the IGP weights for the Rocketfuel topologies, and edge count on the Topology Zoo ones), and random deviations from these (i.e., paths longer by up to 50% with the same end points).

Fig. 5a shows box plots of the measured execution time in function of the path length. As expected, the algorithm exhibits an exponential behavior. Yet, even for longer paths, it still completes in milliseconds: paths of 13 hops have a median runtime of only $\sim 20$ ms. Fig. 5b displays the CDF of the mirroring-rule reduction with respect to mirroring on every hop in the input path (i.e., $1 - \frac{\text{output}}{\text{input}}$). We see that $\sim 80\%$ of the experiments resulted in a gain of over 30%. KPS returned only 2 to 4 mirroring rules in most of the experiments.

**Surrounding algorithm (§4.2)** We run the surrounding algorithm in similar experiments as above, except that for each topology, we randomly select connected components as regions to monitor, and 25% of the nodes having 2 or less outgoing edges as egress points.

Fig. 5c shows the measured execution times, in function of the region size. We observe that: *(i)* computing node surrounding runs in hundreds of microseconds, and is an order of magnitude faster than the further optimized placement; and *(ii)* execution times do not depend on the input region but rather on the network size and its average node degree. Fig. 5d shows the measured optimization gains with respect to edge surrounding. Both algorithms reduce the number of mirroring locations by at least 30% in half of the experiments, and the optimal one can provide an extra gain of 20%.



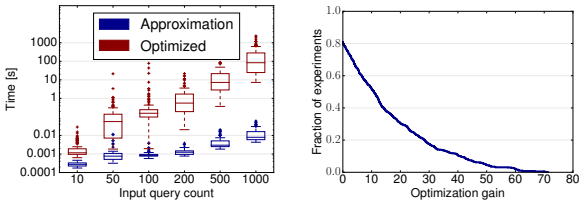

(a) The key-points sampling algorithm runs quickly, even for longer input paths in large graphs.

(b) The key-points sampling algorithm reduced the number of mirroring rules in all experiments.

(c) The surrounding algorithm running times depend on the size of the graph and not the input region.

(d) The surrounding algorithm reduce greatly the number of mirroring rules.

**Figure 5: Computing mirroring rule locations is fast, and the total number of mirroring rules can be drastically reduced, minimizing the mirroring cost for each query and enabling to increase the overall accuracy.**




(a) The scheduling pipeline can produce an approximation at a time-scale suited for online recomputations, even on larger inputs.

(b) By increasing the total number of slot allocations, the optimized schedule maximizes the accuracy of the measurement campaign.

**Figure 6: Stroboscope can compute a quick approximated schedule, or one that maximizes accuracy.**

## 7.2 Scheduling performance

We now evaluate the scalability of our scheduling pipeline on an increasing number of queries. We select a random, normally distributed cost for each query. We vary the time budget for all queries between 20 and 400 timeslots (corresponding to 10 s and 500 ms respectively), and the maximal bandwidth usage per slot between 2 to 100 times the average query cost. We use 10 of those selections per query size.

Fig. 6a shows the running times of the approximated and optimized scheduling algorithm. We confirm that the approximated schedule can indeed be used for online events, as it is computed in microseconds, even for 1,000 queries. The large variance of the optimized pipeline is due to the variation of the maximal bandwidth usage across experiments. If this value is low, it increases the estimated upper bound for the bin-packing problem, which makes computing an optimized schedule exponentially slower. The optimized schedule, however, leads to improved accuracy. Fig. 6b shows the CDF of the relative increase of slot allocation (number of times a query is scheduled in a timeslot), when using the optimized pipeline instead of the approximation. For about half of the experiments the optimized schedule contains 15% more slot allocations than the approximated one, up to 40% for 10% of the experiments.

## 7.3 Real routers mirroring performance

We now present experimental measures on two physical routers (Cisco C7018). Each router mirrors packets to Stroboscope. We connect a traffic generator on the first router, and send test traffic towards the IP address of Stroboscope which is connected to the second router.

**Slice size** We first measure the minimal achievable traffic slice (i.e., activating and immediately deactivating the underlying ACL) and estimate the precision with which we could control the slice duration (by delaying the deactivation of the ACL). Fig. 7 shows the measured duration of the traffic slices depending on the deactivation delay. Each experiment is repeated 50 times. The *minimal slice duration* is $23-25\,ms$. We verify that we precisely control the duration of the traffic slice as it linearly increases with the deactivation delay.

**Mirroring delay** We then measure the time needed by routers to mirror packets by computing the delay between the arrival time of the original and the mirrored packet. The mean mirroring delay over roughly 100,000 measurements is $\mu = 2.6\,\mu s$, with a standard deviation of $\sigma = 1.6\,\mu s$. Such small values indicate that routers mirror packets in constant time.

## 7.4 Reaction to unexpected traffic volume

Finally, we experimentally validate the ability of Stroboscope to react to unexpected traffic increases. In an emulated environment, we configured Stroboscope to mirror a flow of 1 Mb/s at two locations, using at most 5 Mb/s. We then evaluate the ability of Stroboscope to quickly adapt traffic mirroring during a sudden throughput increase. We configured the time during which recorded peak values are used for traffic estimations to 5 seconds.

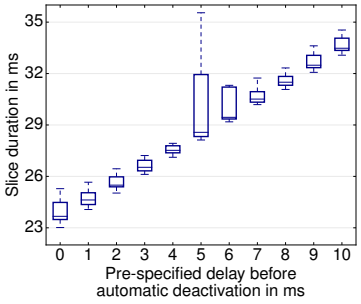



**Figure 7: Existing Cisco routers (C7018) have a minimal slice duration of 23 ms.**

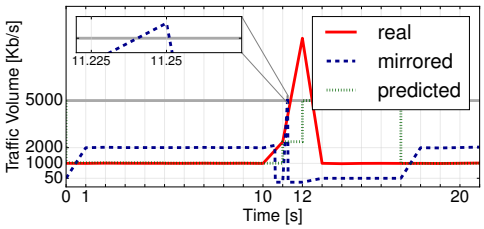



**Figure 8: Stroboscope dynamically estimates traffic demands and swiftly reacts upon budget violation.**

Fig. 8 shows the evolution of *(i)* the real traffic volume of the monitored flow; *(ii)* its predicted traffic demand; and *(iii)* the volume of mirrored traffic. Initially, the prediction starts at the budget value, causing little mirrored traffic as Stroboscope performs the estimation described in §3. After 1 s, the prediction is updated to reflect the last observed peak demand. This increases the amount of mirrored traffic as the query is scheduled more often. At $t = 10\,s$, the real traffic volume spikes, increasing the mirrored traffic. Eventually, the mirrored traffic exceeds the predicted volume, and measurements are interrupted. Traffic prediction is then updated. The same happens after $t = 11\,s$, where the query exceeds the monitoring budget during one timeslot. This causes the traffic estimation from $t = 12\,s$ to $t = 17s$ to be the whole budget, scheduling the query in a single timeslot. In total, the mirrored traffic exceeded the budget for 25 ms.

## 8 Case study: Monitoring transit traffic

We implemented three monitoring applications building upon the measurement stream provided by Stroboscope. Namely, we run our Stroboscope implementation on the queries from Fig. 2 and attach it to router $U$ in the emulated network. Each link has a delay of 5 ms and a loss probability of 1%. We stress that the flow towards 1.2.3.0/24 in the example can be *any* flow—even a tiny one, extremely unlikely to be captured by NetFlow.

**Estimating loss rates** Stroboscope can estimate losses over paths by combining `MIRROR` and `CONFINE` queries. Indeed, there are only three reasons causing a packet captured at the ingress of a path ($A$) to not have a matching copy at the egress ($D$): *(i)* the timeslot completed before the packet reached the egress, which only happens if no packet afterwards is seen at both $A$ and $D$; *(ii)* the `CONFINE` query detected a violation; or *(iii)* the packet was dropped. Using this information, we estimated loss rates across *[A ->D]* to be 7%—slightly higher than the real value (5%) as some mirrored packets were also lost.

**Estimating load-balancing ratios** ECMP hash function polarization [21] causes suboptimal network usage and is hard to detect. We confirmed that Stroboscope can detect such issues by computing a load-balancing ratio: in this setup, the ratio of matching packets seen at $\{A,B,C\}$ over those seen only at $\{A,C\}$, which should be close to 50%. In our case, the monitored prefix had a single flow causing the computed ratio to be about 90% (recall that there are losses in the network). This unusual ratio should prompt operators to observe the captured packet headers.

**Estimating one-way delays** First, Stroboscope estimates router-to-collector latencies. For that, each router has a mirroring rule matching its own loopback address. The collector sends probes towards these loopbacks, and receives copies echoed by the NIC (with no CPU fallback on the routers). Stroboscope finds the router-to-collector latency by comparing the probe and echo timestamps. Second, Stroboscope estimates one-way delays between routers ($A$ and $D$) by: *(i)* identifying matching packets in their traffic slices; *(ii)* reconstructing the time at which the packets traversed each router by subtracting the router-to-collector latency from the time at which the mirrored packet copy was received at the collector; *(iii)* computing the difference between these traversal times. Using this procedure, we confirmed that the latency of *[A ->D]* was 15 ms. Note that this estimation does not require to use any form of clock synchronization between the routers and the collector.

## 9 Related Work

**Stream-based monitoring** Stroboscope relates to Gigascope [24], a stream-based system which provides a SQL-like query language to stream packet-based measurements from any router interface. In contrast to Stroboscope, Gigascope lacks higher-level constructs such as path-based queries and the ability to adhere to a monitoring budget. It also supports fewer concurrent queries as changing the packet dissectors they execute on the routers is slow.

| Feature | Stroboscope | Everflow [22] | Planck [23] |
|---|---|---|---|
| Query-based mirroring | ✓ | ✓ | ✗ |
| Monitoring on a budget | ✓ | ✗ | ✗ |
| Runs on commodity hardware | ✓ | ✓ | ✓ |
| Independence from active probing | ✓ | ✗ | ✓ |
| Independence from header bits | ✓ | ✗ | ✓ |

**Table 1: Comparison between Stroboscope and other mirroring-based techniques.**

**Mirroring-based monitoring** Stroboscope is not the first system to use packet mirroring for monitoring purposes. For example, [11] relies on packet mirroring to selectively monitor control-plane traffic. In Table 1, we compare Stroboscope with Everflow [22] and Planck [23], the two mirroring-based systems which are the closest to Stroboscope. Only Stroboscope can comply with a mirroring budget. Also, Stroboscope does not require active packet marking or special header flags as Everflow's [22] "guided probe" approach does.

**Monitoring with programmable hardware** Progress in programmable hardware (e.g., P4 [25]) and virtual network devices (e.g., Open vSwitch [26]) enables new monitoring possibilities. SketchVisor [27] is a sketch-based measurement framework built on virtual switches. Basat et al. [28] present a randomized constant time algorithm to identify hierarchical heavy hitters. NetQRE [29] uses regular expressions over packet streams to express flow-level and application-level policies. All three approaches could directly be built on top of Stroboscope. Other works focus on compiling high-level queries into specific actions of programmable devices. In [30, 31], path queries are supported by encoding the path traversed by packets in the packets header. Narayana et al. [32] introduce a performance query language, Marple, interacting with a key-value store running on the switches. By scheduling mirroring rules network-wide, Stroboscope supports path or Marple queries without the need for rewriting packets or special network data structures. More generally, our work shows that hardware capabilities of current routers are sufficient to build programmable monitoring systems.

**Monitoring flow statistics** Tools like NetFlow [1] are often used in ISP networks and provide coarse-grained flow statistics by randomly sampling traffic. FlowRadar [33] and ProgME [34] provide per-flow packet counters. While they can also bound the monitoring overhead, these approaches lack the capability of Stroboscope to track individual packets across the network, and thus cannot measure fine-grained statistics such as one-way delays or load-balancing ratios.

**Data-center monitoring** Many research contributions on network monitoring provide fine-grained traffic visibility in settings different from ISPs, mainly data centers. They exploit degrees of freedom that are unavailable in ISP networks, especially control of end-hosts, e.g., to collect fine-grained statistics [35] or probe the network [36]. Stroboscope is a more general in-network solution, viable in any network, *including ISP ones*. We note that some Stroboscope building blocks can be useful in other settings as well. For example, its internal algorithms could be used in a new version of Everflow [22] which keeps the mirrored traffic volume under control.

**Network Verification** Stroboscope complements recent initiatives in data-plane [37, 38, 30, 39, 40, 41] and control-plane [42, 43, 44, 45] verification by enabling dynamic testing of runtime-based predicates such as performance metrics (e.g., measuring packet loss). Stroboscope similarly complements recent efforts for building debugging tools for software defined networks [46, 47].

## 10 Conclusions

As networks grow in complexity, they require flexible monitoring tools able to measure precise metrics about their traffic flows while scaling to ever-growing traffic volumes. In this paper, we show how Stroboscope achieves these objectives by combining the visibility benefits of traffic mirroring with the scalability of traffic sampling. Specifically, Stroboscope enables to collect fined-grained measurements of any traffic flow while adhering to a monitoring budget. Stroboscope works with existing routers, and is well-suited for ISP networks. We believe that Stroboscope monitoring capabilities could address the visibility needs of many future network applications, including self-driving network control loops.

## Acknowledgements

We are grateful to NSDI anonymous reviewers, our shepherd Boon Thau Loo, Lynne Salameh and Roland Meier for their insightful comments. O. Tilmans is supported by a grant from F.R.S.-FNRS FRIA. This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 688421, and was supported by the Swiss State Secretariat for Education, Research and Innovation (SERI) under contract number 15.0268. The opinions expressed and arguments employed reflect only the authors' views. The European Commission is not responsible for any use that may be made of that information. Further, the opinions expressed and arguments employed herein do not necessarily reflect the official views of the Swiss Government.

## References


[1] Benoit Claise. Cisco Systems NetFlow Services Export Version 9. RFC 3954 (Informational), October 2004. http://www.ietf.org/rfc/rfc3954.txt.

[2] Peter Phaal, Sonia Panchen, and Neil McKee. InMon Corporation's sFlow: A Method for Monitoring Traffic in Switched and Routed Networks. RFC 3176 (Informational), September 2001. http://www.ietf.org/rfc/rfc3176.txt.

[3] Yin Zhang, Sumeet Singh, Subhabrata Sen, Nick Duffield, and Carsten Lund. Online identification of hierarchical heavy hitters: algorithms, evaluation, and applications. In *SIGCOMM*, pages 101–114, 2004.

[4] Olivier Tilmans, Tobias Bühler, Stefano Vissicchio, and Laurent Vanbever. Mille-Feuille: Putting ISP Traffic Under the Scalpel. In *HotNets*, pages 113–119, 2016.

[5] Jon Postel. Internet protocol darpa internet program protocol specification. RFC 791, September 1981. https://tools.ietf.org/rfc/rfc791.txt.

[6] Puneet Agarwal and Bora Akyol. Time To Live (TTL) Processing in Multi-Protocol Label Switching (MPLS) Networks. RFC 3443, January 2003. https://tools.ietf.org/rfc/rfc3443.txt.

[7] Jianer Chen, Yang Liu, and Songjian Lu. An improved parameterized algorithm for the minimum node multiway cut problem. *Algorithmica*, 55:1–13, 2009.

[8] György Dósa. The tight bound of first fit decreasing bin-packing algorithm is ffd (i) $\leq$ 11/9 opt (i)+ 6/9. *Combinatorics, Algorithms, Probabilistic and Experimental Methodologies*, pages 1–11, 2007.

[9] Cisco Systems. Configuring ERSPAN, 2016. https://goo.gl/h3qaGL.

[10] Juniper Networks. Layer 2 Port Mirroring Overview, 2014. https://goo.gl/YxgZuY.

[11] Stefano Vissicchio, Luca Vergantini, Luca Cittadini, Valerio Mezzapesa, Maurizio Pizzonia, and Maria Luisa Papagni. Beyond the Best: Real-time Non-invasive Collection of BGP Messages. In *INM/WREN*, 2010.

[12] Cisco Python API. http://bit.ly/2fMgyKP.

[13] Junos Automation Scripts Overview, 2017. https://goo.gl/WpjAcX.

[14] Pedro Marques, Nischal Sheth, Robert Raszuk, Barry Greene, Jared Mauch, and Danny McPherson. Dissemination of Flow Specification Rules. RFC 5575 (Proposed Standard), August 2009. http://www.ietf.org/rfc/rfc5575.txt.

[15] R. Enns et al. NETCONF Configuration Protocol. RFC 4741, December 2006. https://tools.ietf.org/rfc/rfc4741.txt.

[16] Stefano Vissicchio, Olivier Tilmans, Laurent Vanbever, and Jennifer Rexford. Central Control Over Distributed Routing. In *SIGCOMM*, pages 43–56, 2015.

[17] A. Bobyshev, P. DeMar, and D. Lamore. Effect of dynamic ACL (access control list) loading on performance of Cisco routers. In *Computing in High Energy Physics*, 2004.

[18] Mininet: An Instant Virtual Network on your Laptop (or other PC). 2012. http://www.mininet.org/.

[19] Neil Spring, Ratul Mahajan, and David Wetherall. Measuring ISP Topologies with Rocketfuel. In *SIGCOMM*, pages 133–145, 2002.

[20] Simon Knight, Hung X. Nguyen, Nickolas Falkner, Rhys Bowden, and Matthew Roughan. The internet topology zoo. *IEEE Journal on Selected Areas in Communications*, 29:1765 –1775, 2011.

[21] Cisco Tech Support. CEF Polarization. 2013. https://goo.gl/b7ZSMy.

[22] Yibo Zhu, Nanxi Kang, Jiaxin Cao, Albert Greenberg, Guohan Lu, Ratul Mahajan, Dave Maltz, Lihua Yuan, Ming Zhang, Ben Y. Zhao, and Haitao Zheng. Packet-Level Telemetry in Large Datacenter Networks. In *SIGCOMM*, pages 479–491, 2015.

[23] Jeff Rasley, Brent Stephens, Colin Dixon, Eric Rozner, Wes Felter, Kanak Agarwal, John Carter, and Rodrigo Fonseca. Planck: Millisecond-scale Monitoring and Control for Commodity Networks. In *SIGCOMM*, pages 407–418, 2014.

[24] Chuck Cranor, Theodore Johnson, Oliver Spataschek, and Vladislav Shkapenyuk. Gigascope: a stream database for network applications. In *SIGMOD*, pages 647–651, 2003.

[25] Pat Bosshart, Dan Daly, Glen Gibb, Martin Izzard, Nick McKeown, Jennifer Rexford, Cole Schlesinger, Dan Talayco, Amin Vahdat, George Varghese, and David Walker. P4: Programming Protocol-independent Packet Processors. *SIGCOMM Comput. Commun. Rev.*, 44:87–95, 2014.

[26] Ben Pfaff, Justin Pettit, Teemu Koponen, Ethan J Jackson, Andy Zhou, Jarno Rajahalme, Jesse Gross, Alex Wang, Joe Stringer, Pravin Shelar, et al. The Design and Implementation of Open vSwitch. In *NSDI*, pages 117–130, 2015.

[27] Qun Huang, Xin Jin, Patrick P. C. Lee, Runhui Li, Lu Tang, Yi-Chao Chen, and Gong Zhang. SketchVisor: Robust Network Measurement for Software Packet Processing. In *SIGCOMM*, pages 113–126, 2017.

[28] Ran Ben Basat, Gil Einziger, Roy Friedman, Marcelo C. Luizelli, and Erez Waisbard. Constant Time Updates in Hierarchical Heavy Hitters. In *SIGCOMM*, pages 127–140, 2017.

[29] Yifei Yuan, Dong Lin, Ankit Mishra, Sajal Marwaha, Rajeev Alur, and Boon Thau Loo. Quantitative Network Monitoring with NetQRE. In *SIGCOMM*, pages 99–112, 2017.

[30] Peyman Kazemian, Michael Chang, Hongyi Zeng, George Varghese, Nick McKeown, and Scott Whyte. Real Time Network Policy Checking Using Header Space Analysis. In *NSDI*, pages 99–111, 2013.

[31] Srinivas Narayana, Mina Tahmasbi, Jennifer Rexford, and David Walker. Compiling Path Queries. In *NSDI*, pages 207–222, 2016.

[32] Srinivas Narayana, Anirudh Sivaraman, Vikram Nathan, Prateesh Goyal, Venkat Arun, Mohammad Alizadeh, Vimalkumar Jeyakumar, and Changhoon Kim. Language-Directed Hardware Design for Network Performance Monitoring. In *SIGCOMM*, pages 85–98, 2017.

[33] Yuliang Li, Rui Miao, Changhoon Kim, and Minlan Yu. FlowRadar: A Better NetFlow for Data Centers. In *NSDI*, pages 311–324, 2016.

[34] Lihua Yuan, Chen-Nee Chuah, and Prasant Mohapatra. Progme: towards programmable network measurement. *IEEE/ACM Transactions on Networking (TON)*, 19:115–128, 2011.

[35] Masoud Moshref, Minlan Yu, Ramesh Govindan, and Amin Vahdat. Trumpet: Timely and Precise Triggers in Data Centers. In *SIGCOMM*, pages 129–143, 2016.

[36] Chuanxiong Guo, Lihua Yuan, Dong Xiang, Yingnong Dang, Ray Huang, Dave Maltz, Zhaoyi Liu, Vin Wang, Bin Pang, Hua Chen, Zhi-Wei Lin, and Varugis Kurien. Pingmesh: A Large-Scale System for Data Center Network Latency Measurement and Analysis. In *SIGCOMM*, pages 139–152, 2015.

[37] Haohui Mai, Ahmed Khurshid, Rachit Agarwal, Matthew Caesar, P. Brighten Godfrey, and Samuel Talmadge King. Debugging the Data Plane with Anteater. *SIGCOMM Comput. Commun. Rev.*, 41:290–301, 2011.

[38] Peyman Kazemian, George Varghese, and Nick McKeown. Header Space Analysis: Static Checking for Networks. In *NSDI*, pages 113–126, 2012.

[39] Ahmed Khurshid, Xuan Zou, Wenxuan Zhou, Matthew Caesar, and P. Brighten Godfrey. VeriFlow: Verifying Network-Wide Invariants in Real Time. In *NSDI*, pages 49–54, 2013.

[40] Nuno P Lopes, Nikolaj Bjørner, Patrice Godefroid, Karthick Jayaraman, and George Varghese. Checking Beliefs in Dynamic Networks. In *NSDI*, pages 499–512, 2015.

[41] Radu Stoenescu, Matei Popovici, Lorina Negreanu, and Costin Raiciu. SymNet: Scalable Symbolic Execution for Modern Networks. In *SIGCOMM*, pages 314–327, 2016.

[42] Ari Fogel, Stanley Fung, Luis Pedrosa, Meg Walraed-Sullivan, Ramesh Govindan, Ratul Mahajan, and Todd Millstein. A General Approach to Network Configuration Analysis. In *NSDI*, pages 469–483, 2015.

[43] Aaron Gember-Jacobson, Raajay Viswanathan, Aditya Akella, and Ratul Mahajan. Fast Control Plane Analysis Using an Abstract Representation. In *SIGCOMM*, pages 300–313, 2016.

[44] Konstantin Weitz, Doug Woos, Emina Torlak, Michael D. Ernst, Arvind Krishnamurthy, and Zachary Tatlock. Scalable Verification of Border Gateway Protocol Configurations with an SMT Solver. *ACM SIGPLAN Notices*, 51:765–780, 2016.

[45] Ryan Beckett, Aarti Gupta, Ratul Mahajan, and David Walker. A General Approach to Network Configuration Verification. In *SIGCOMM*, pages 155–168, 2017.

[46] Andreas Wundsam, Dan Levin, Srini Seetharaman, and Anja Feldmann. OFRewind: Enabling Record and Replay Troubleshooting for Networks. In *USENIX ATC*, pages 15–17, 2011.

[47] Nikhil Handigol, Brandon Heller, Vimalkumar Jeyakumar, David Mazières, and Nick McKeown. I Know What Your Packet Did Last Hop: Using Packet Histories to Troubleshoot Networks. In *NSDI*, pages 71–85, 2014.

# A Proofs for the placements algorithms

## A.1 Proof for Theorem 1

**Theorem.** *Let a forwarding path $P$ be the concatenation of sub-paths $Q_1, \ldots, Q_n$.* **`MIRROR`** *queries on $P$ can be correctly answered by mirroring only on the endpoints $s_i$ and $t_i$ of all $Q_i$ such that no other forwarding path from $s_i$ to $t_i$ has the same length as $Q_i$.*

*Proof.* First, we show that if any $Q \subseteq P$ is the only path of length $x$ from $p$ to $s$, we can always distinguish mirrored packets that have been forwarded over $Q$ by just mirroring on $p$ and $s$. Let $l$ be the length of $Q$, and let $t_p$ and $t_s$ be the TTL values of any packet mirrored from $p$ and $s$. Under the assumption that the TTL is properly decreased (by one) at each forwarding hop, we can unambiguously determine if the packet has been forwarded over $Q$: If $t_s = t_p - l$, then $Q$ must be the traversed sub-path because $Q$ is the only path of length $l$ between $p$ and $s$ by hypothesis.

The statement of the theorem then follows by noting that the same property applies to any sub-path $Q_i$, as well as to their concatenation—i.e., $P$. □

## A.2 Proof for Theorem 2

**Theorem.** **`CONFINE`** *queries on a region $R$ can be correctly answered if and only if the set of mirroring locations is the edge surrounding of $R$.*

The following lemma proves Theorem 2.

**Lemma 1.** *Given a region $R$, its edge surrounding $E(R)$ and any prefix $d$, it is guaranteed to capture all and only packets for $d$ that exit $R$ if and only if mirroring rules matching $d$ are active on every edge in $E(R)$.*

*Proof.* We first show that all packets exiting $R$ are captured if and only if rules are placed on all the edges in $S(R)$. Consider any packet $p$ entering from an ingress node $i$ in $R$. For $p$ to exit $R$, there must be a node $r$ in $R$ (possibly $r = i$) that forwards $p$ to a node $o$ outside $R$. The packet will then traverse the edge $(r,o)$ with $r \in R$ and $o \notin R$. If mirroring rules are active on all the edges in $S(R)$, then $p$ is detected, by definition of $S(R)$. In contrast, if a mirroring rule is not active on any edge $(r_m, o_1)$ with $r_m \in R$ and $o_1 \notin R$, then a packet $p'$ will not be mirrored if it exits $R$ through $(r_m, o_1)$ and never enters $R$ again – e.g., following a path $[r_1 \ldots r_m o_1 \ldots o_k]$, where $\forall i = 1, \ldots, m\ r_i \in R$ and $\forall j = 1, \ldots, k\ o_j \notin R$.

In addition, if a packet is captured by a rule placed on an edge $(x,y)$ in $S(R)$, then it must have crossed a node $x$ in $R$ and be forwarded to a node $y$ outside $R$, by definition of $S(R)$. This implies that only packets leaving $R$ are mirrored, which yields the statement. □

## A.3 Proof for Theorem 3

**Theorem.** *In the absence of forwarding anomalies, a* **`CONFINE`** *query on a region $R$ can be correctly answered if and only if every mixed-egress path for $R$ contains at least one mirroring location.*

The following lemma proves Theorem 3.

**Lemma 2.** *In the absence of forwarding anomalies, any packet not confined to a region $R$ is guaranteed to be mirrored if and only if every simple path starting from a node in $R$, traversing a node outside $R$ and ending in any egress point crosses at least one active mirroring rule matching the packet destination.*

*Proof.* We separately prove sufficiency and necessity of the condition expressed by the theorem.

*Sufficiency:* Proof by contradiction. Assume that some packets not confined to $R$ are not mirrored despite mirroring rules matching the condition in the theorem statement. In the absence of forwarding anomalies, those packets are guaranteed to be delivered to an egress point. Not to be confined to $R$, they must follow a path $[r_1 \ldots r_n o_1 \ldots o_l \ldots e]$, where $e$ is an egress point, nodes $r_i \in R\ \forall i = 1, \ldots, n$, and nodes $o_j \notin R\ \forall j = 1, \ldots, l$. By hypothesis, an active mirroring rule must be on this path and must mirror the packets, contradicting the assumption that packets are not mirrored.

*Necessity:* Proof by contradiction. Assume that it is guaranteed to mirror all packets confined to $R$ but no mirroring rule is active on a given path $P = [r_m \ldots o \ldots e]$ from a node $r_m \in R$ to an egress point $e$ including a node $o \notin R$ (possibly $o = e$). Consider now any path $[r_1 \ldots r_m]$, where $m \geq 1$, $r_1$ is an ingress point, and $r_i \in R$ $\forall i = 1, \ldots, m$. This path must exist since a region is defined as a connected component (see §2). Packets forwarded on the concatenation of the previous two paths (i.e., $[r_0 \ldots r_m \ldots o \ldots e]$) are not confined to $R$, as they cross $o \notin R$. However, they are not mirrored, contradicting the assumption. □

## B Scheduling ILP formulations

### B.1 Optimal bin-packing

**Input** All queries and their associated costs, an upper bound on the number of of timeslots needed.

**Decision Variables** Let $Q$ be the set of input queries and $S$ the set of all time slots in the measurement campaign. We define:

$R_{qs}$ as the binary variable representing the decision to schedule the query $q \in Q$ in timeslot $s \in S$ when $R_{qs} = 1$;

$U_s$ as the binary variable representing whether the timeslot $s \in S$ has any assigned query when $U_s = 1$.

**Parameters**

$B$ The maximal available bandwidth in a single timeslot;

$a_q$ The expected traffic volume generated by the mirroring rules for the query $q$.

**Objective Function** Minimize the length of the subschedule

$$\min \sum_{s}^{S} U_s$$

**Constraints**

C1 In any timeslot $s$, the expected traffic generated by the mirroring rules across all queries activated in $s$ must be lesser or equal than the budget.

$$\forall s : \sum_{q}^{Q} (R_{qs} a_q) \leq U_s B$$

C2 Every query must be scheduled.

$$\forall q : \sum_{s}^{S} R_{qs} = 1$$

C3 Track used slots.

$$\forall q,\ s : U_s \geq R_{qs}$$

C4 Timeslots should be used in sequence (tie-breaking constraint).

$$\forall s,\ s',\ s < s' : U_s \leq S_{s'}$$

### B.2 Maximal filling

**Input** A list of queries and their associated cost, a list of time slot and leftover budget. Queries are pruned such that any query whose cost is greater than the biggest leftover budget available is excluded.

**Decision Variables** Let $Q$ be the set of input queries and $S$ the set of all time slots in the measurement campaign. We define:

$R_{qs}$ as the binary variable representing the decision to schedule the query $q \in Q$ in timeslot $s \in S$ when $R_{qs} = 1$;

$M$ as the continuous variable representing the minimal number of slots allocated to any query.

**Parameters**

$\beta_s$ The available leftover bandwidth in the timeslot $s$, thus $\beta_s \leq B$;

$\Omega$ The spreading factor, which lets the operator favor schedules where all queries have a similar number of timeslots (high value) or schedules maximizing the absolute number of allocation;

$w_q$ The preference level of the query $q$. Queries with a higher preference are scheduled preferably to queries with a lower preference;

$a_q$ The expected traffic volume generated by the mirroring rules for the query $q$.

**Objective Function** Maximize the utilization of the budget, either by maximizing the number of allocations of some queries, according to their preference level, or by spreading the budget across all queries (thus maximizing the minimal allocation).

$$\max \left[ \sum_{q}^{Q} \left( \sum_{s}^{S} R_{qs} \right) w_q + M\Omega \right]$$

**Constraints**

C1 In any timeslot $s$, the expected traffic generated by the mirroring rules across all queries activated in $s$ must be lesser or equal than the leftover budget.

$$\forall s : \sum_{q}^{Q} (R_{qs} a_q) \leq U_s \beta_s$$

C2 $M$ should represent the minimal number of allocated slots across all queries.

$$\forall q : M \leq \sum_{s}^{S} R_{qs}$$